A Quick Guide on Training a neural network using Keras.

TensorFlow and Keras

Keras

- Open source
- High level, less flexible
- Easy to learn
- Perfect for quick implementations
- Starts by François Chollet from a project and developed by lots of people.
- Written in Python, wrapper for Theano, TensorFlow, and CNTK

TensorFlow

- Open Source
- Low level, you can do everything!
- Complete documentation
- Deep learning research, complex networks
- Was developed by the Google Brain team
- Written mostly in C++ and CUDA and Python

https://keras.io/

https://www.tensorflow.org/

Keras

Feed Forward MNIST

Recurrent networks, cosine function

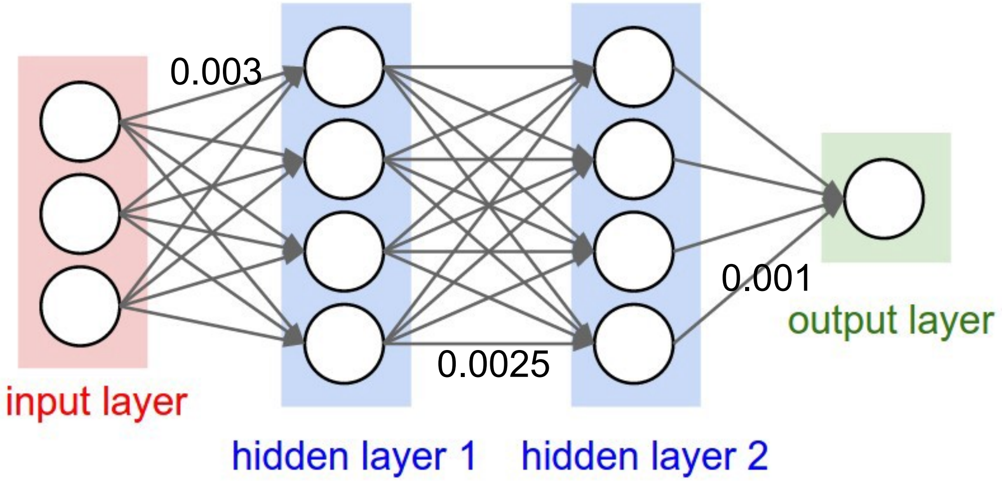
Convolutional MNIST

https://en.wikipedia.org/wiki/MNIST_database

Train Feed-Forward Network

...

2.5
4.1
3.0

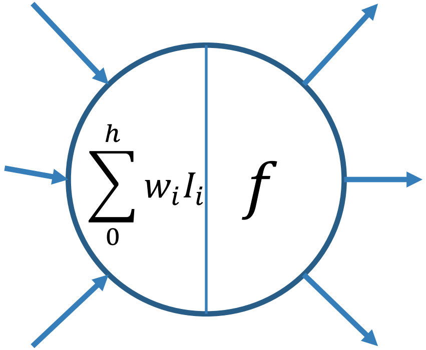


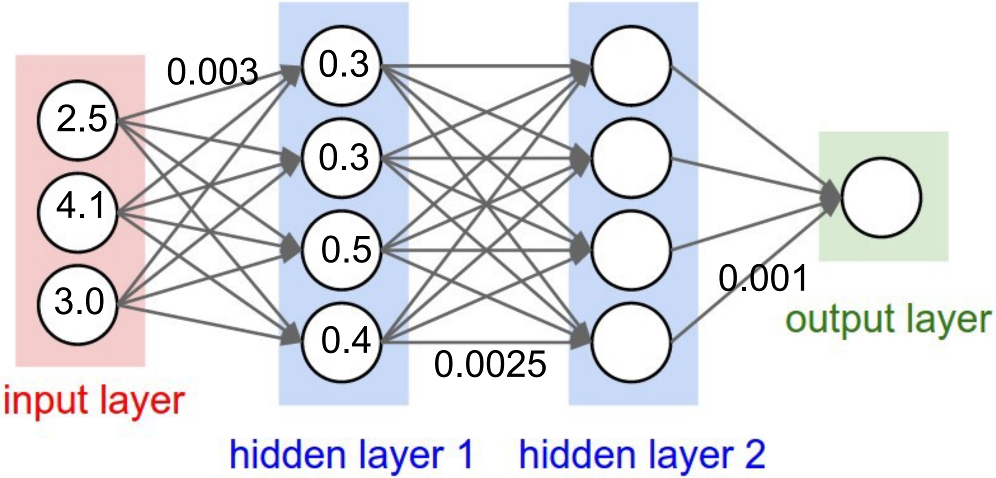
Target: 1.0

Weight Initialization and Loading the Data

More on Weight Initialization: http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

Train Feed-Forward Network



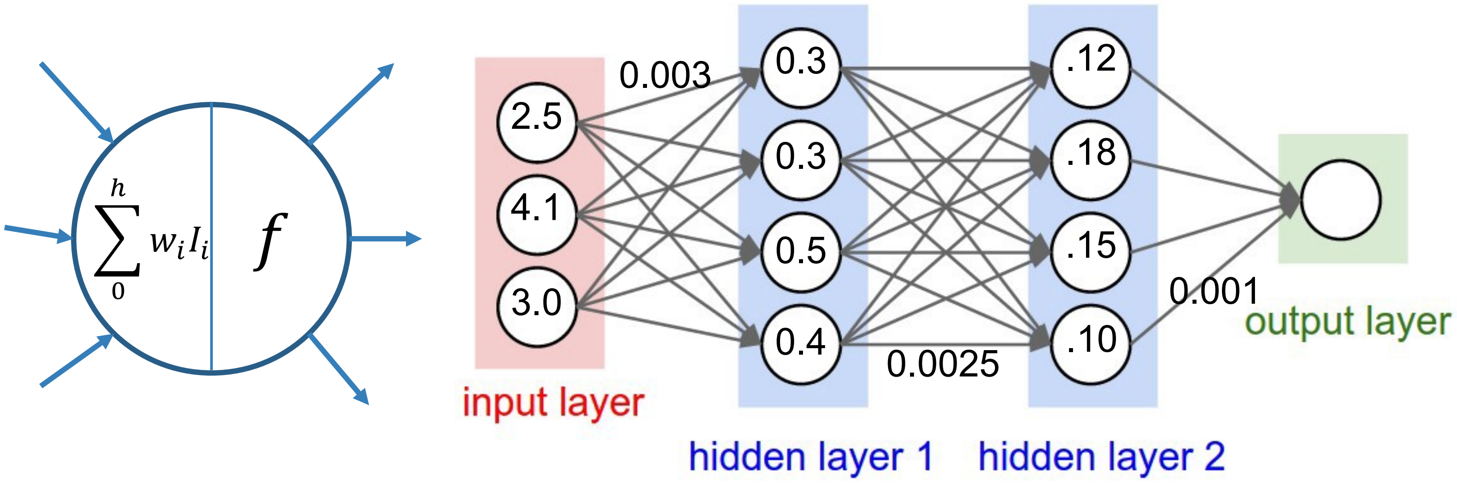


Target: 1.0

Going forward (Regularization, Dropout, Batch Normalization, …)

More on Dropout: http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

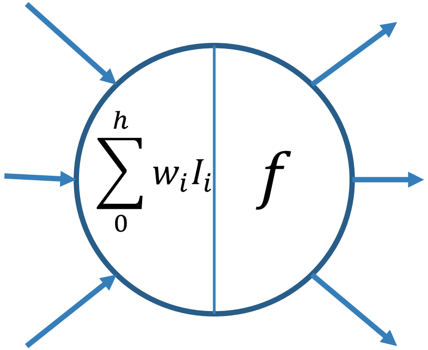
Train Feed-Forward Network

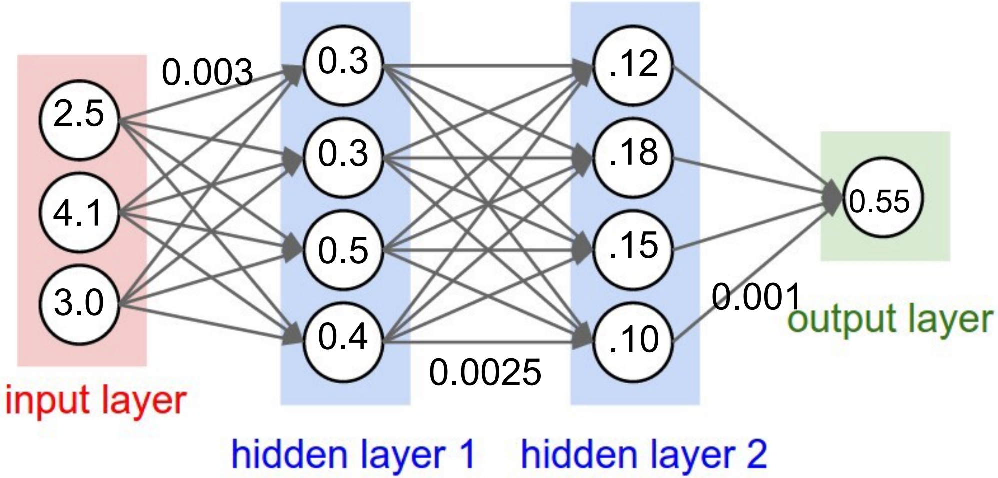


Target: 1.0

Going forward (Regularization, Dropout, Batch Normalization, …)

Train Feed-Forward Network



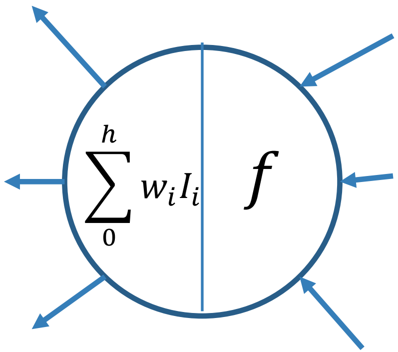


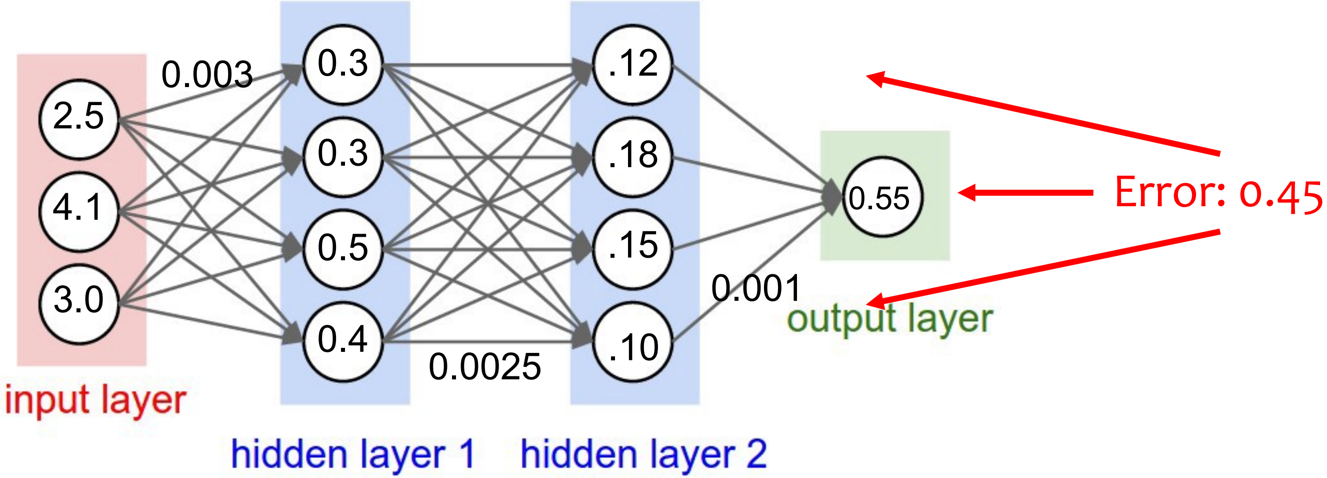
Target: 1.0

$$Loss\ Function: (T - O) \Rightarrow Loss: 0.45$$

More on Loss Function: https://arxiv.org/abs/1702.05659

Train Feed-Forward Network



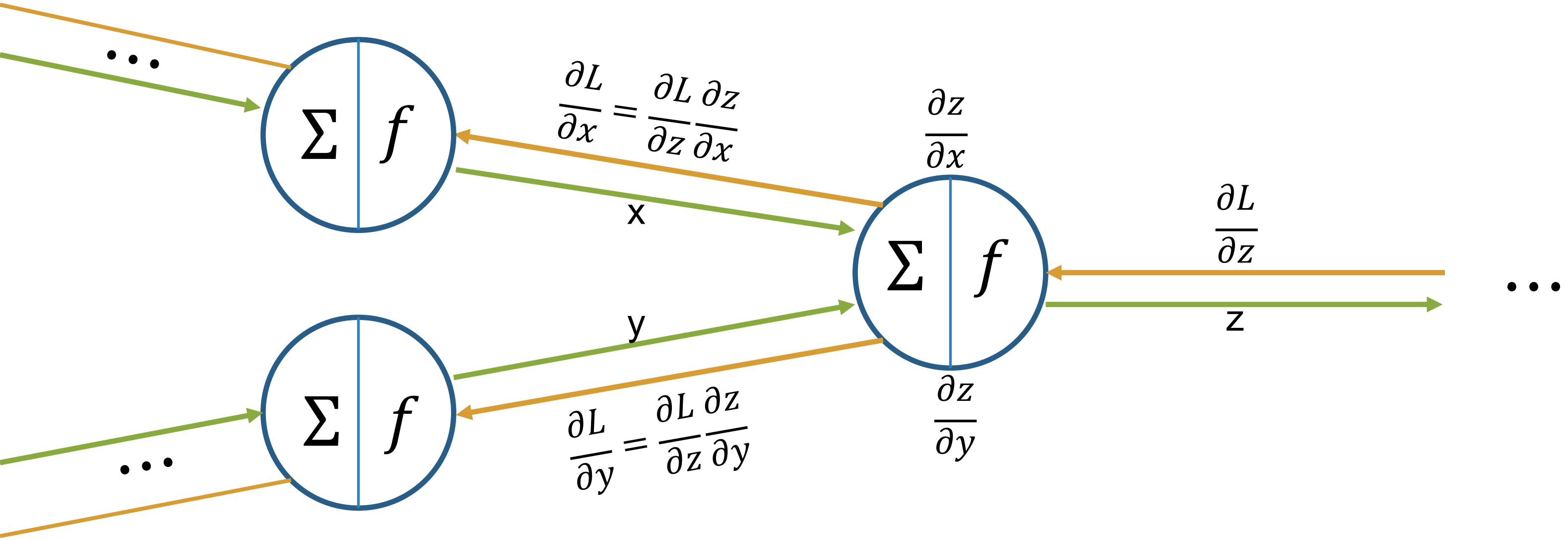


Updating the weights using Backpropagation $\frac{\partial E}{\partial w_j^i}$

More on Backpropagation: http://cs231n.github.io/optimization-2/

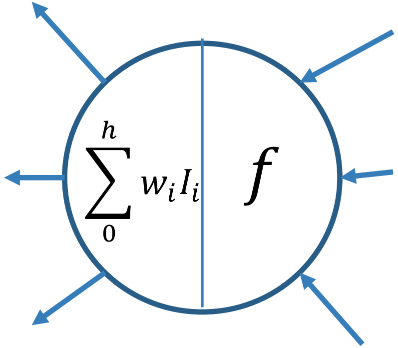
http://arxiv.org/abs/1502.05767

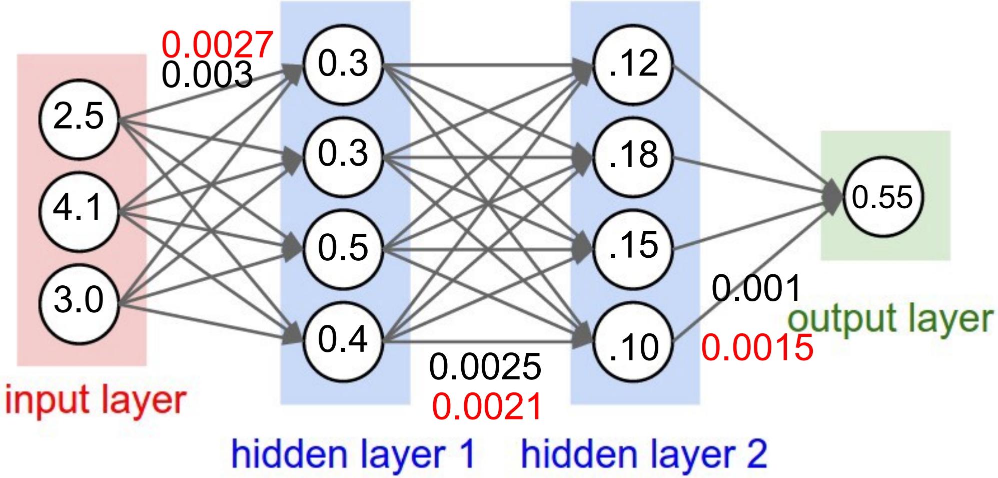
Train Feed-Forward Network



Backpropagation

Train Feed-Forward Network





By Choosing the Optimizer, new weights will be computed $w_{new} = w - \eta \frac{\partial E}{\partial w}$

A quick explanation on Optimizers: [Optimizers]

Now Let's Code!

Design the network Architecture

- **Sequential Model**
- Add layers
- Define all operations
- Define the output layer

Instantiate an object from Keras.model.Sequential class. All information about your network such as weights, layers, operations will be stored in this object.

Design the network Architecture

- Sequential Model
- **Add layers**
- Define all operations
- Define the output layer

The magic ”add” method, makes your life easier!
You can create any layer of your choice in the network using model.add(layer_name).
This method will preserve the order of layers you add.

Design the network Architecture

- Sequential Model
- Add layers
- Define operations per layer
- Define the output layer

There are lots of layers implemented in keras. When you add a layer to you model, a gradient operation will be created in the background and it will take care of computing the backward gradient automatically!

Design the network Architecture

Sequential Model

Add layers

Define all operations

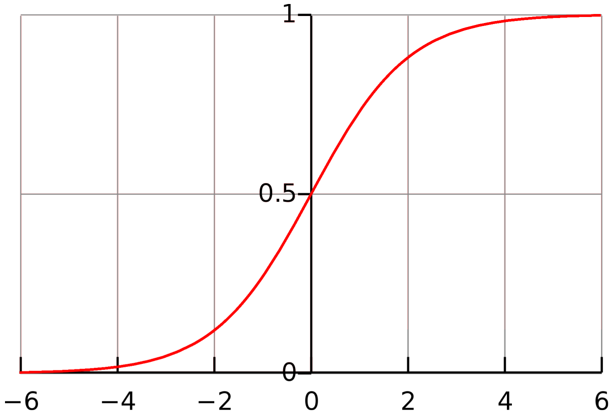
Define the output layer

In this example we use Dense layer, which is the basic feed forward fully connected layer.
All operations of a layer can be passed as args to the Dense Object.

1. Number of hidden units
2. Activation function
3. Bias
4. Weight/bias initialization
5. Weight/bias regularization
6. Activation regularization

Activation Function

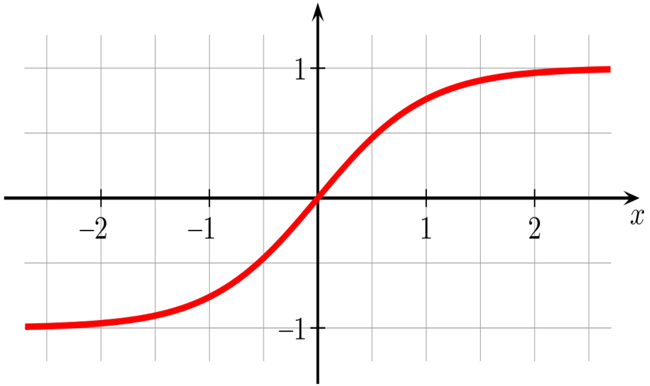
$$sigmoid: f(x) = \frac{1}{1 + e^{-x}}$$



- The output is not zero centered!
- Exp() is a relatively expensive operation.
- Small active region

Activation Function

$tanh: f(x) = \tanh(x)$



- Out put is zero centered (good)
- Exp() is an expensive operation
- Smaller active region!

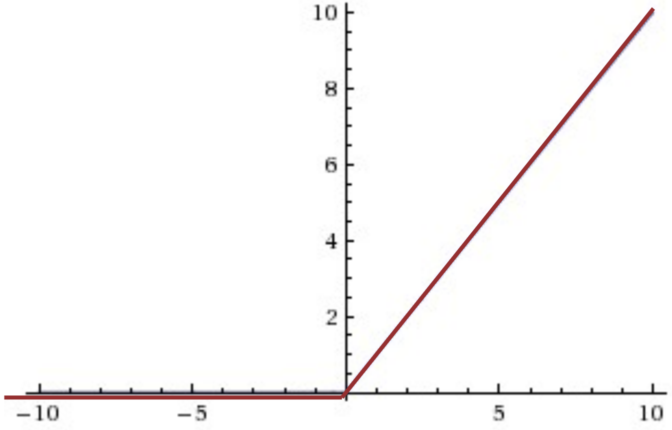
Activation Function

Sigmoid() is not a good choices!

Think about the backward path when all then inputs to a neuron are positive numbers.

Activation Function

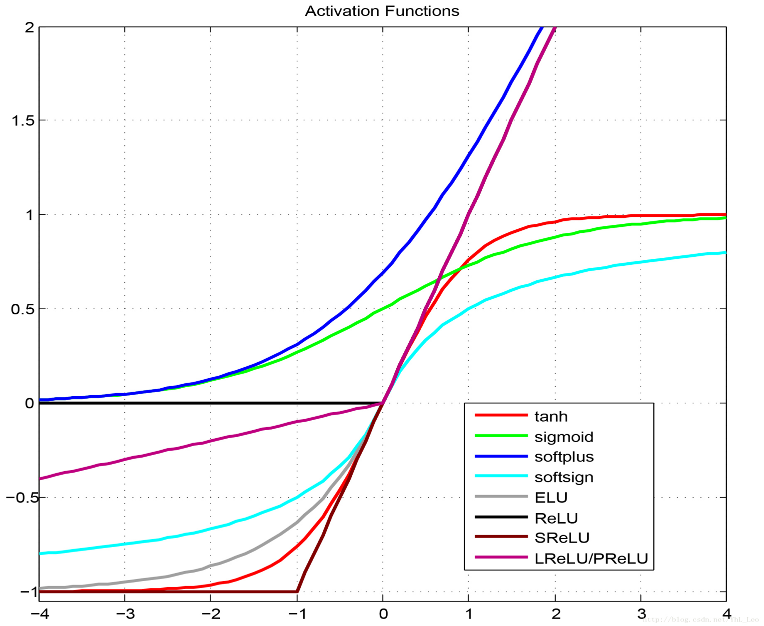
$$ReLU\ (Rectifier\ Linear\ Unit): f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



A neuron will be active during the backprop if the out put was a positive number.

Activation Function

Other activation functions



Weight initialization

Initialize all the Ws at Zero!

Small random numbers

Xavier Initialization

Other Ideas

Weight initialization

What happen when you initialize all the weights at Zero?

Hint: Think about backpropagation

Weight initialization

Small random numbers?

0.001*np.random.randn(M, N)

Not good for deep networks!
What happened during the backpropagation?

Weight initialization

Xavier Initialization
[Glorot et al., 2010]

```
np.random.randn(fan_in, fan_out) / np.sqrt(fan_in)
```

Fixes the problem of small random numbers with linear activation.

Weight initialization

You can use other available initializations e.g. [He et al., 2015]
Or you can write your own initialization.

- Exact solutions to the nonlinear dynamics of learning in deep linear neural networks
- Random walk initialization for training very deep feedforward networks
- Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification
- Data-dependent Initializations of Convolutional Neural Networks
- All you need is a good init

Regularization

L1 Regularization

$$loss \mathrel{+}= \lambda\left(\sum_{0}^{h} |w_i|\right)$$

L2 Regularization

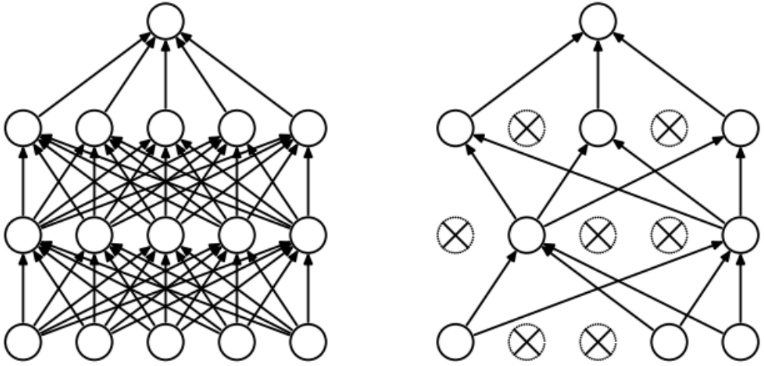
$$loss \mathrel{+}= \lambda\left(\sum_{0}^{h} w_i^2\right)$$

Sanity Check: your loss should become larger when you use regularization.

Dropout

`keras.layers.Dropout(rate)`

Randomly setting a fraction rate of input units to 0 at each update during training time.



(a) Standard Neural Net

(b) After applying dropout.

More about Dropout: [Dropout]

Design the network Architecture

Sequential Model

Add layers

Define all operations

Define the output layer

Based on the task of prediction, you need to define your output layer properly.
For classification task on MNIST dataset, we have ten possible classes, so it’s a multiclass classification.
So it’s better to use softmax activation for a ten unit output layer.

Compile your model

```
model.compile(loss, optimizer, metrics)
```

Losses: mean_squared_error, squared_hinge, categorical_hinge, binary_crossentropy, …

Optimizers: SGD, Adam, RMSprop, ...

Metrics: binary_accuracy, categorical_accuracy, top_k_categorical_accuracy

Loss Function

Categorical crossentropy is the appropriate loss function for the *softmax* output

For linear outputs use *mean_squared_error*

For *logistic*outputs use *binomial crossentropy*

Some notes on the math behind the loss: [Peter's note]

Optimizers

```
while True:
    batch_of_data = training_set.sample_data_batch()
    loss = NN.forward_path()
    dw = NN.backward_path()
    w += -learning_rate * dx
```

Optimizers

```
#Momentum update
v = mu * v - learning_rate * dw #think of v as velocity.
w += v
```

```
#Nestrov momentum update
v_previous = v
v = mu * v -learning_rate * dw
w += -mu * v_previous + (1+mu) * v
```

Optimizers

```
#Adagrad update
epsilon = 1e-7
mem += dw**2
w += -learning_rate * dw / (np.sqrt(mem) + epsilon)
```

```
#RMSProp
epsilon = 1e-7
mem = decay_rate * mem + (1 - decay_rate) * (dw**2)
w += -learning_rate * dw / (np.sqrt(mem) + epsilon)
```

Train the model

History = model.fit(X_train, Y_train, batch_size, …)

Batch_size

Epoches

Validation split, Validation data

Callbacks

Shuffle

Class_weights

Train the model

```
History = model.fit(X_train, Y_train, batch_size, … )
```

History Object

Attribute **History.history** sotres training loss, validation loss, metrics, validation metrics values at successive epochs in a python dictionary.

Evaluate the performance

```
model.evaluate(X_test, Y_test, verbose)
```

Returns a list of score. First element is the loss and the rest are the metrics you specified during the compilation of your model.

Do the prediction!

```
model.predict_classes(data)
```

You can use your trained model to predict the class of a given input.

How can we make it better?

Keras

Feed Forward MNIST

Recurrent networks, Learning cosine function

Convolutional MNIST

More on Recurrent neural networks: [Anrej Karpathy note]
[Colah's blog]

Recurrent Network

- What is the digit written in the image?

- What is the next character?

Hell_

Recurrent Network

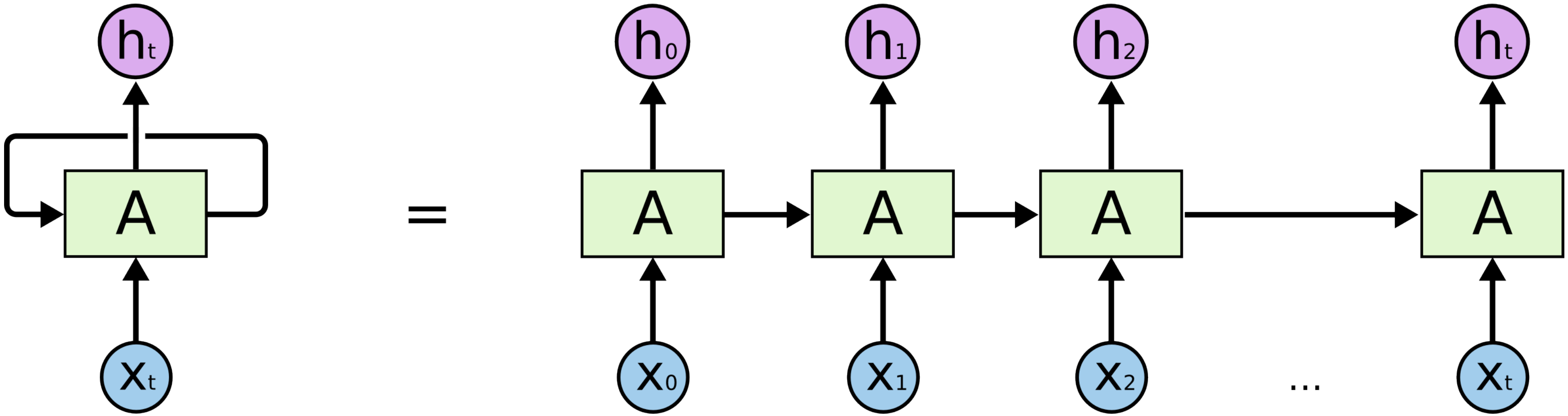
- What is the digit written in the image?

Input: an Image (a matrix of numbers)
Output: one of the 10 possible classes

- What is the next character?

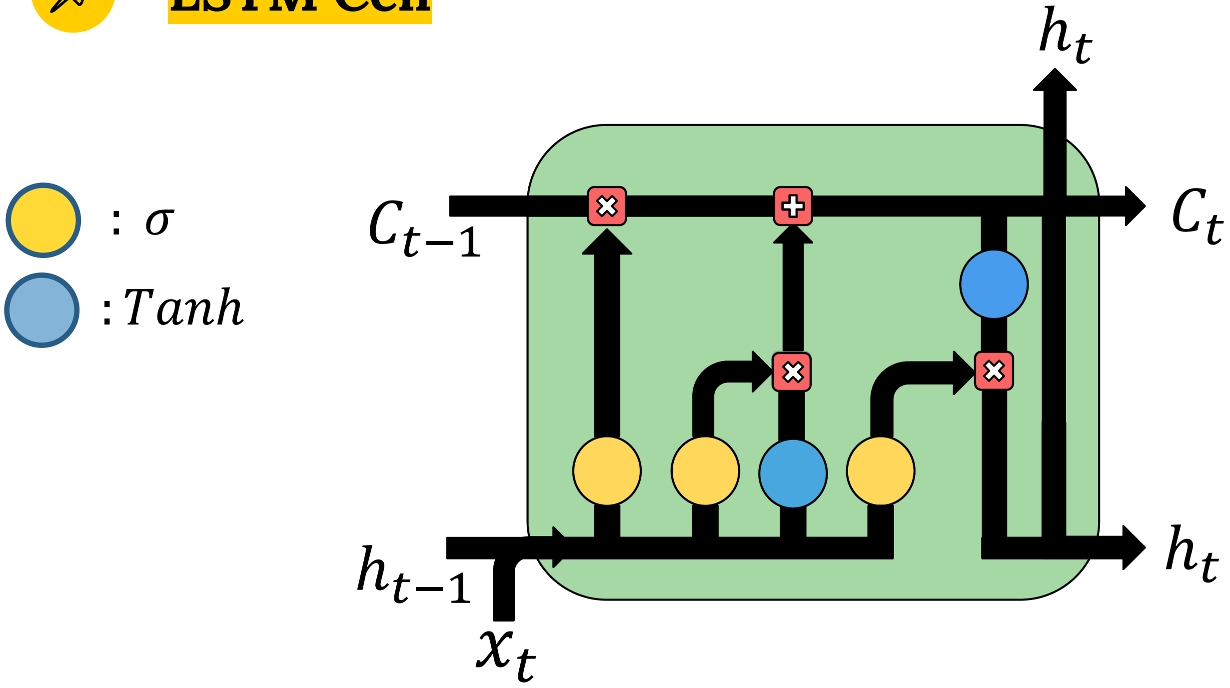
Hell_

Input: ?
Output: one of the 26 possible classes



h_t
A
x_t
=
h_0 h_1 h_2 h_t
A A A A
x_0 x_1 x_2 ... x_t

LSTM Cell



Learn the cosine function by looking at previous inputs.

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

Wider Window

Stacked LSTM

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

Wider Window

Stacked LSTM

How can we make it better?

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

Wider Window

Stacked LSTM

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

How can we make it better?

Wider Window

Stacked LSTM

Recurrent Network, LSTMs

- Vanilla LSTM
- Stateful LSTM
- **Wider Window**
- Stacked LSTM

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

Wider Window

Stacked LSTM

How can we make it better?

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

Wider Window

Stacked LSTM

Recurrent Network, LSTMs

Vanilla LSTM

Stateful LSTM

Wider Window

Stacked LSTM

Let’s try a feed forward network!